Amendments to the Specification:

Pursuant to 37 C.F.R. § 1.121(b) kindly amend the specification as follows. Amendments to the specification are made by presenting replacement paragraphs or sections marked up to show changes made relative to the immediate prior version. The changes in any amended paragraph or section are being shown by strikethrough (for deleted matter) or underlined (for added matter).

Page 1, Line 1

A METHOD AND APPARATUS FOR PERFORMING IMAGE PROCESS OF SEISMIC DATA

Page 3, Lines 18-23 Delete paragraph.

Page 4, lines 13-18

Another aspect of the present invention is directed to an apparatus for generating an image of a mass. A plurality of sources being located in the mass. The apparatus comprises memory configured to store information defining energy levels recorded at a sensor, the energy levels emanating from one or more of the sources. A data processor has circuitry. An algorithm has a plurality of tasks for determining the level of energy emanating from each source and recorded at the sensor. The algorithm is implemented in the circuitry of a data processor.

Page 4-5, lines 19-2

Another aspect of the preset invention is a method of manufacturing a circuit for generating an image of a mass. A plurality of sources ~~is~~ are located in the mass wherein energy is emanating from the sources. The image is generated from the energy levels recorded at a sensor. The method comprises creating a gate-level netlist, the netlist corresponding to an algorithm having a plurality of tasks for determining the energy level emanating from each source and recorded at the sensor; and placing and routing the netlist in circuitry.

Page 14, lines 14-19

The DFU 208 and the parallel execution module 202 are depicted as overlapping to illustrate their relationship. As will be described more fully below, selected ~~ones of the~~ functional execution units 206 are used in connection with the DFU208 to create control structures corresponding to the control flow of the algorithm. The resulting control structures are therefore dictated by the particular algorithm to which the architecture is directed.

Page16, lines 11-17

The integrated memory control within the DFU 208 controls the data flow from the bulk memory bank 204 to the DFU 208. ~~The~~ This invention allows for the elimination of cache memory by distributing processors, and obtaining cache-like performance using the bulk memory 204 as controlled by the integrated memory control. An address/control path 214 couples the integrated memory controller to the bulk memory bank 204, and transmits the appropriate address and control signals required to obtain the desired memory output. The memory control is further described in connection with FIG. 3.

Pages 17-18, lines 1-2

The example DFU 300 includes a memory control module 302 and a module 304 for realization of the functional algorithm. The memory control module 302 is designed in hardware such that all address and data control signals are derived and transmitted in a single clock cycle. The memory control module 302 includes an address cluster 306, shown in Figure 3 and a data control cluster 308. ~~While the~~ The particular implementation of the address cluster 306 is dependent on the functional algorithm being implemented, and therefore on the particular manner in which data must be read from the bulk memory bank 204~~, FIG. 3 provides a representative address cluster 306~~. The address cluster 306 includes address sequencing functionality 310, address generation functionality 312, and an address output 314. The address generator 312 generates a starting address for a memory fetch. Because a preferred embodiment of the present invention involves the use of memory block transfers by way of memory "bursts", only one starting address needs be provided for any giving block of data that is desired. The starting address is ~~output~~ outputted to the bulk memory bank 204 via the address output

314, which comprises a dedicated, non-multiplexing data distribution path coupled directly to the address input of the memory 204. The address sequencer 310 recognizes the length of a desired burst of data, and notifies the address generator 312 when the next block of data requiring a new starting address is going to be transmitted. In one embodiment, the address sequencer 310 is set to automatically increment to the next burst starting address when the desired burst count is reached, so that only a clock pulse is required as input to the memory controller. It should be recognized that the address cluster 306 is constructed in hardware such that an address can be ~~output~~ outputted from the address output 314, needing only the receipt of a single clock pulse. Pipelining within the address cluster 306 can be implemented where necessary to insure that a starting address is provided at the address output 314 on the next available clock pulse following transmission of a complete data block.

Page 18-19, lines 3-7

The data control cluster 308 is similarly constructed in-hardware. The control cluster 308 includes a burst length generator 316 which provides control signals to the bulk memory bank 204 via the data control output 318. For memory devices which allow burst transfers of particular lengths, the burst length generator can provide the number of bytes desired. In one embodiment of the invention, burst length generation is not required. In that particular embodiment, an SDRAM capable of performing full-page bursts with a burst terminate option is utilized. In this case, a starting address plus a full-page burst command can be presented to the SDRAM, and the address sequencer 310 thereafter provides a burst terminate command when appropriate. For example, the address generator 312 can output a starting address to the memory 204 along with a command to initiate a burst transfer. A predetermined number of bytes later, the address sequencer 310 can provide a burst terminate command via the address output 314 to the memory 204 to stop the burst transfer after a predetermined number of bytes have been transferred from the memory 204. This memory control configuration eliminates control signal protocols which require additional clock cycles. The memory control 302 is therefore a variable-length memory controller providing variable-length data bursts as required by the algorithm, which may or may not require the use of a potentially deep pipeline. For example, where a 100-byte data burst is provided to the data flow unit to perform the functional algorithm, a 100-register pipeline can be used in the memory control module 302 to ensure timely presentation of a subsequent data burst

starting address immediately upon transmission of the final byte of a preceding memory burst. It should also be recognized that, depending on the format of the raw data in the bulk memory bank 204, the memory control module 302 can be configured to provide variable-length data bursts (i.e., 30 bits in a first burst, 31 in a second burst, etc.). The variable-length burst control can also be varied to accommodate different algorithms. Furthermore, in one embodiment of the invention, "tiling" is implemented to provide data bursts from within a single row of memory to avoid address wrap-around conditions. This allows the "structure" of the memory to be exploited, as opposed to viewing it as a homogeneous bank of storage cells.

Page 19, Lines 12-19

The memory control module 302 therefore becomes a physical replica of the memory control portion of the functional algorithm. The memory access patterns generated by the memory control 302 provide for sustained execution of the DFU 208 and the parallel execution module 202 shown in FIG. 2. This allows all functional execution units within the processing module to be concurrently and continually active. ~~Direct~~ and direct, non-multiplexed distribution paths between the memory control 302 and the memory facilitate single cycle memory control. This configuration results in "cache-like" performance using standard memory.

Page 20, lines 3-12

The algorithm realization module 304 also works in connection with functional execution unit 206 described in connection with FIG. 2. The data path distribution 330 essentially includes the bussing and data ~~pipe lining~~ pipelining required to effect single cycle execution. The bussing architecture within the algorithm realization module 304 includes discrete, dedicated buses to provide the data from the memory bank 204 to the desired one or more functional execution units 206. These buses are non-interleaved, non-multiplexed buses that are dictated by the particular functional algorithm being implemented. Multiplex data buses would require additional clock cycles. ~~Pipe lining~~ Pipelining may also play an important role in the distribution of data depending on the particular algorithm to be implemented. Pipelining is described in greater detail in connection with the description of FIGS. 6A, 6B, and 6C.

Pages 20-21 lines 20-6

FIG. 4 is a block diagram conceptually illustrating a hardware manifestation of a functional algorithm in accordance with the present invention. For purposes of illustration and not of limitation, the functional algorithm 400 of FIG. 4 is depicted as having multiple functions, including function A 402, function B 404, function C 406, function D 408, through function n-1 410 and function n 412. The number of processing functions to be performed is dependent upon the particular algorithm to be applied in hardware. To perform any particular function, multiple sub-functions may be necessary. For example, looking now to function D 408, one particular sub-function is ~~illustration~~ illustrated as sub-function 414. As an example, function D 408 may represent an absolute value function, and the sub-function 414 may represent one of a plurality of inverters used in the particular implementation of an absolute value function.

Page 22, lines 1-13

The memory control module 430, analogous to the memory control module 302 of FIG. 3, performs the functions previously described in connection with FIG. 3. As was previously described, the memory control module 430 ensures a continual stream of data to the data path 428 to be distributed throughout the functional algorithm 400 by way of the bus 426. The functional execution units, sub-functions, functions, control structures, discrete bussing, and pipeline structures of the functional algorithm 400 allow the functional algorithm to be executed in a single clock cycle when the pipeline, if any, is full. Therefore, ~~where~~ software execution of an algorithm that includes ~~is a~~ sequential ~~process~~ processes involving one or more ~~perhaps~~ multiple clock cycles during ~~on~~ certain steps of the algorithm, may be effected ~~the entire functional algorithm can be effected~~ in a single clock cycle with the present invention, including the entire functional algorithm. ~~Where~~ When the functional algorithm is continually repeated, for example, to perform similar computations on large volumes of test data input, new data can be processed on each clock cycle. This substantially increases the rate at which the raw test data can be processed.

Pages 23-24, lines 17-2

A_1 and B_1 are ~~input~~ inputted into an exclusive-OR (XOR) module 512 which includes an OR-gate 514, a NAND-gate 516 and an AND-gate 518 to produce the sum on output line 520. Where both A_1 and B_1 have binary 1 values, AND-gate 522 produces a carry signal on line 524 to the

next level execution unit (not shown). As will be described in further detail below, the bussing architecture in one embodiment of the present invention utilizes non-multiplexed, non-interleaved, dedicated signal paths to facilitate single-cycle execution. Where each of the instructions of the inner loop are similarly converted to parallel execution units, it can be seen that the entire functional algorithm embodied by the inner loop can be realized in hardware.

Page 24, lines 3-10

The multiple execution units aggregately reflect the functional algorithm, and in fact are dictated by the algorithm to execute in a single clock cycle. Inner loops and other core software routines associated with data-intensive applications are typically quite short, thereby making hardware implementation of the algorithm practicable. The particular algorithm of the present invention dictates the content of the multiple execution units, as well as their interconnection. In some instances, the interconnection is effected through the use of a potentially deep data pipeline to account for timing considerations, as described in connection with FIGS. 6A, 6B 10 and 6C below.

Page 25, lines 13-18

Algorithm sub-function 606 is performed by one of the functional execution units 610 in the system. Execution unit 610 receives as input variables A and B, and produces the first SUM1 iteration labeled $SUM1_{,IN}$ on path 612. Input variables C and D are also concurrently inputted ~~input~~. FIG. 6C illustrates this timing, as variables A, B, C and D are provided at the triggering edge of clock pulse 614, and $SUM1_{IN}$ on path 612 occurs during the same clock pulse, slightly delayed due to propagation delays (propagation delay not illustrated).

Pages 26-27, lines 3-2

The $SUM1_{IN}$ signal on path 612 is provided to a hardware control structure 624 which performs the do-loop 608. Various sub-functions are performed as part of the do-loop 608, including an addition sub-function (SUM1=SUMI+C), and a comparison sub-function to determine whether the loop has been executed the requisite number of times. Because the variable C changes on each of the three clock cycles (i.e., three consecutive memory cycles provide three different values of C), this sub-function requires three clock cycles to completely

execute. This is illustrated in FIG. 6C, where the variable C changes on each of the triggering clock pulses 626, 628 and 630. Also, on the occurrence of each of the clock pulses 626, 628 and 630, the variable D propagates through the pipeline 616. At clock pulse 614, the variable D is passed through register-1 618 (stage-1), which in turn is passed through register-2 620 on clock pulse 628 (stage-2), and finally ~~passed~~ passes through register-3 622 on clock pulse 630 (stage-3). The $SUM1_{OUT}$ signal on path 632 is not provided to the execution unit 636 until the triggering edge of clock pulse 630 occurs, due to the three-stage do-loop 608. Similarly, the variable D is not provided on path 634 to the execution unit 636 until the triggering edge of clock pulse 630 occurs, due to the three-stage pipeline 616. Therefore, the $SUM1_{OUT}$ signal on path 632 and the staged variable D on path 634 reach the execution unit 636 during the same clock period, and SUM2 is executed and produced at the output of execution unit 636 on path 638. Once the pipeline 616 is full, a new SUM2 will be produced on each occurrence of a clock signal. Similar executions are carried out for each portion of the algorithm, ultimately resulting in an output of an entire algorithm loop on each clock cycle. Once full, the pipeline remains full, and is as deep as required to produce an algorithm output on each clock cycle.

Page 31, lines 8-10

At this point, hardware implementations for the algorithm and each of the individual tasks within the algorithm (i.e., sub-functions) are derived 1104. To illustrate, the nested loop 1008 ~~is analyzed~~ is analyzed in connection with FIGS. 12-17.

Page 33, lines 1-7

FIG. 15 illustrates one approach to transforming a software mathematical function into hardware. The statement SUM = SUM + INDATA(I) can be transformed to hardware as ~~illlustrated~~ illustrated. The INDATA(I) on input path 1500 is provided to register 1502, and inputted ~~input~~ to the adder 1504 when the clock signal on path 1506 activates the register 1502. The adder output is staged at register 1508, the output of which is the SUM. The current value of SUM is fed back into the adder to accomplish the desired addition function. This addition takes only one pipeline stage.

Pages 35-36, lines 17-5

Concurrent with the memory controller operation, the data is distributed via discrete transmission paths to the data flow unit and parallel execution units, as seen at block 1808. The data is preferably distributed in a non-multiplexed, non-interleaved fashion to facilitate single-cycle execution. Data dependencies dictated by the expression evaluation and control flow of the algorithm are accounted for using a fully-pipelined distribution path. Again, this facilitates single-cycle execution of the algorithm. Once the pipeline is full, one algorithm iteration is ~~output~~ outputted from the processing module on each clock cycle, as indicated at block 1810. Where the algorithm does not require pipelining, it can be seen that an algorithm output would also be provided in each clock cycle, the difference being that the data output from the memory would immediately be acted upon and ~~output~~ outputted (i.e., data would not be "staged" throughout the data flow unit and parallel execution module). Subsequent algorithm outputs are provided upon the occurrence of subsequent ones of the clock pulses 1802.

Page 40, lines 12-23

FIG. 22 illustrates a ~~one-clock-cycle~~ one clock cycle implementation of an absolute value function. This embodiment takes advantage of special information about the particular data formats employed. For example, in 2's complement format, the expression (A < 0) reduces to the value of the most significant bit (MSB) or "sign bit". Where the MSB is binary "0" ~~0~~, A is positive, and the input value of A is selected by the multiplexer 2200 as designated by the sign bit from the register 2202. Where the MSB is binary ~~1~~ "1", it indicates that A is a negative number, requiring inversion 2204 of the remaining bits of A to produce the absolute value. Therefore the sign bit of "A" can be used to control the multiplexer 2200 to select between "A" or "not A" for every bit except the sign bit, which is hardwired to binary "0". The selected value is stored in register 2206. Provided that the delays through the inverters 2204 and the multiplexer 2200 are less than one clock cycle, a ~~single-cycle~~ single cycle implementation of abs(a) is realized.

Page 41, lines 1-6

It should be recognized that the particular implementation (i.e., the ~~two-clock-cycle~~ two clock cycle or the ~~one-clock-cycle~~ one clock cycle implementation) is essentially irrelevant due to the unique, fully-pipelined architecture of the present invention. This is because once the

pipeline has been filled, an entire algorithm output is provided, regardless of how deep the pipeline is within the architecture. An insignificant and virtually undetectable time difference is present, however, during pipeline filling and emptying, due to extraneous staging registers.

Page 43, lines 8-17

Each such external memory component is controlled by the memory controller to supply a word of data (or accept a word of data for writing) in every clock cycle. Thus, the processing clock is held to be the same as the memory bus data rate. To prevent data bus conflicts and facilitate single-cycle execution, each memory bank has its own data, address and control bus. This can be seen in FIG. 25, which illustrates a data flow unit (DFU) 2500 coupled to receive information from multiple SDRAM banks 2502, 2504 and 2506. Each of the SDRAM banks includes a dedicated data, address, and control bus. For example, SDRAM bank 2502 includes data bus 2508, address bus 2510 and control bus 2512, which are distinct and separate from data bus 2514, address bus 2516 and control bus 2518 associated with SDRAM 2506.

Page 48, lines 10-17

This architecture has significant advantages for imaging. In particular, the migration algorithm for moving an observed event back to its true spatial or temporal position can be transformed into hardware through the transfer rules provided. In the case of migration, one possibility is determining the amount of energy that emanates from a point within a mass. This energy forms a point in an image. The image as a whole represents the internal structure of the mass that has been analyzed. Once the emanated energy for each point is displayed, the structure of the mass becomes determinable and may be inputted ~~input~~ into a graphical image if desired.

Page 53, lines 12-16

Once the travel time generator has generated all of the travel time volumes, the migration process can begin. The travel time volume and the trace data are ~~input~~ inputted into the migration processor 308 employing the architecture discussed herein. The migration processor 308 then extracts from the trace data the total amount of energy that emanated from each point in the subsurface.

Page 56, lines 5-20

The travel time processor 1000 is used to interpolate a travel time for a point in the image volume. The travel time processor 1000 can be a field programmable gate array (FPGA) or an application specific integrated circuit (ASIC). The clock rate for an FPGA can typically be set at 50-66 MHz. The clock rate can be increased to 100-200 MHz for an ASIC. ~~One designs the~~ The algorithm is designed in a design software such as VHDL. This allows one compiler to create executable software of the design which is useful for verification ~~but which~~ and is running under a simulator at approximately one millionth of the speed at which the FPGA or ASIC will operate. The other compiler, called a synthesis tool, produces gate level netlists which are fed into a place and route tool which places and routes the architecture design and then outputs a bitstream that is downloaded into the FPGA. To create the ASIC, the gate level netlist is sent to an ASIC manufacturer who then uses industry standard techniques to place and route the netlist into the chip. Before the netlist is sent to the manufacturer a gate level simulation and/or timing verification is done. The gate level netlist is a description at the lowest level, i.e. individual logic gate level, of a hardware embodiment of the design. The term"to place" ~~To place~~ means to assign an individual physical gate on the chip to each gate described in the netlist. The term"to route" ~~To route~~ means to interconnect the gates so placed in accordance with the netlist.

Page 59, lines 1-5

With reference to Fig. 36, the I/O phase and the migration phase of the migration process can be described at the top level. During the I/O phase trace and travel time, volume data ~~are~~ is loaded through the PCI bus and into the travel time processor 1000 (some buses used for loading data are not shown). The travel time volume is directed to the travel

Page 62, lines 1-10

The image volume memory is updated by a ping pong effect. All implementations of the ping pong effect can be described in terms of ~~the~~ two modes. The following describes one implementation. In one mode, two words per clock cycle are read from the image volume memory 1010. One is modified by the sum to image operator 1106 and both words are written to cache 1008. After completing the current output trace, which is typically 2000 points in depth,

the next mode begins. In ~~this other~~ the second mode, the two words come from cache into the sum to image operator 1106 and another new amplitude value is added to the previously unmodified word. The updated words representing accumulated values are then sent back to the image volume I/O 1108 which sends them to the appropriate address in the image volume memory 1010. This occurs for all 2000 points in depth.

Page 64, lines 18-23

The travel time unpack unit 1300 accepts single precision floating point travel time values and scales them to 24 bit integers. The travel time unpack 1300 receives the ~~a~~ pair travel time values for 2 adjacent points along the y axis that are at the same x and z location as indicated by the x,y,z incrementers. This pair of travel times is received from the travel time volume memory data path 1200 through a unidirectional data bus capable of carrying multiple words.

Page 71, lines 12-22

The next clock cycle provides the add-pass operator 1526 with all of the data it needs to generate the interpolated output. It receives the product contained in the first register of the third stage 1518 through a unidirectional data bus. It receives the flag indicating whether the travel times are of value from the second register in the third stage 1520 through a unidirectional data bus. It also receives the product contained in the third register of the third stage 1522 and the signal indicating whether the fraction is zero ~~through~~ contained in the fourth register of the third stage 1524 through unidirectional data buses. If the flag indicates that the fraction is not zero, then the add-pass operator 1526 adds the two products. If the flags indicates that the fraction is zero, then the add-pass operator 1526 has received the travel time originally stored in register 1500 via multiplier 1510 and register 1518 since the multiplier merely multiplied this travel time by one.